

more on operator overloading

Monday, October 18, 2010
9:00 AM

- using operator += to define operator +

```
vec_t v4, v5;
double d = 6.0;
```

v4 += v5

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}_{v4} + \begin{pmatrix} 6 \\ 2 \\ 1 \end{pmatrix}_{v5} = \begin{pmatrix} 7 \\ 4 \\ 4 \end{pmatrix}_{v4}$$

v4 += d;

same for v4 *= d;

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}_{v4} + \begin{pmatrix} d \\ d \\ d \end{pmatrix}$$

same as old vec_scale(a, b, c)

// compound assignment operator +=, from:

// <http://www.cs.caltech.edu/courses/cs11/material/cpp/donnice/cpp-ops.html>

vec_t & operator += (const vec_t & rhs)

```
{
  if (this != &rhs) // not sure if this is needed - prob. not
  for (int i = 0; i < 3; i++) vec[i] += rhs[i]
  return *this;
}
```

will this work for a += a? should work...

? (a+b) = c
allow 3 why? if don't know
const vec_t operator + (--- rhs)

Const

→ allow 3 why! if don't know

vec_t operator + (--- rhs)

{

vec_t result (*this); \equiv vec_t result = *this,

result += rhs;

return(result)

}



can reduce further

const vec_t operator + (const vec_t & rhs)

{

return vec_t(*this) += rhs;

}

material_t

```
class material_t  
{  
    :  
    private :  
    int cookie ;  
    std::string name ;  
    dir_t aphabet ;  
    " diff  
    " speaker ;  
};
```

- up at the top:

// constructor

material_t () : \

```
cookie (MAT_COOKIE), \
ambient (0.0, 0.0, 0.0), \
:
};
```

// copy constructor

// destructor (default ok)

// argument errors

// friends

- - - - -

- - - - -

class
↑
interface

```

material_t class implementation!
}
#include <iostream>
#include <string>
using namespace std;
"prod.h"
"material.h"

```

```

std::ostream& operator<<(std::ostream& s, const material_t& rhs)
{

```

```

s<<"material_ "<< rhs.name.c_str() << std::endl;

```

```

s<<"{" << std::endl; // gives C-style char*

```

```

(s<<"{\n";)

```

```

if(!rhs.ambient.is_zero()) s<<"_ ambient " << rhs.ambient << std::endl;

```

same for diffus, specular

```

s<<"}" << std::endl << std::endl;

```

```

return s;
}

```

```

std::istream& operator>>(std::istream& s, material_t& rhs)
{

```

```

char c;
std::string allname;

```

// have read "material" token.

```

s>> rhs.name;

```

// consume all char until '{'

```

while(s.good() && s.get(c) && (c!='{'));

```

// loop until we hit '}'

```

while((c = s.peek()) != '}') {

```

// read attr name

```

s>> allname;
}

```

```
// read in attribute & consume whitespace at EOL  
if (attrname == "attribute") s >> rhs.attribute >> std::ws;
```

```
"  
save for diffise, operator
```

```
} // end of while
```

```
// eat '}' character
```

```
while (s.peek() && s.get() && (c != '}'));
```

```
return s;
```

```
}
```

material_t* material_getbyname (list_t& mats, std::string name) ^{material name fn ↑ looks for}

{
 material_t * mat;

 mats.reset();

 while (mats.not_end()) {

 mat = (material_t*) mats.get_data();

 assert (mat->getcookie() == MAT_COOKIE);

 ↑ have to provide the
 accessor for
 otherwise compiler
 says:

 mat->cookie

 is private

 if (mat->getname() == name) return mat;

 mats.next_link();

}

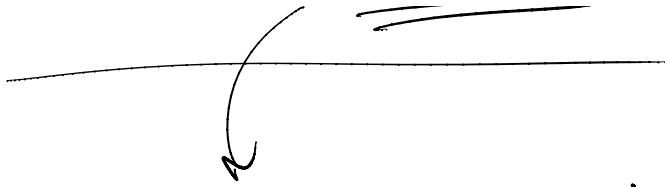
return NULL;

}

in 212, we use

C++ built-in lists ("containers")

with iterators



```
std::list<material_t * > mats;
```

```
std::list<material_t * >::iterator it;
```

```
while (it = mats.begin(); it < mats.end(); it++)
```

```
{
```

```
if ((*it) -> getname() == name)
```

```
return (*it)
```

```
}
```

```
return null;
```